Adaptive Peer-to-peer Routing with Proximity

Chu Yee Liau[1], Achmad Nizar Hidayanto[2], and Stephane Bressan[1]

[1] School of Computing,
Department of Computer Science,
National University of Singapore
{liaucy, steph}@comp.nus.edu.sg
[2] Faculty of Computer Science,
University of Indonesia
nizar@cs.ui.ac.id

Abstract. In this paper, we presented a routing strategy for requests in unstructured peer-to-peer networks. The strategy is based on the adaptive routing Q-routing. The strategy uses reinforcement learning to estimate the cost of routing a request. Such a strategy is scalable only if the routing indices are of reasonable size. We proposed and comparatively evaluated three methods for the pruning for the pruning of the routing indices. Our experiments confirm the validity of the adaptive routing and the scalability of a pruning approach based on a pruning strategy considering the popularity of the resources.

1 Introduction

Peer-to-peer networks constitute an architectural revolution that may, if the managerial and economical conditions are met, replace the traditional client server approaches to information management. Potentially, peer-to-peer may solve the scalability issues as raised by the advent of a global information infrastructure. This infrastructure has millions of users exchanging large amounts of information from computers of all sorts and kinds across the Internet for all kinds of purpose ranging from leisure to business and government.

Yet, peer-to-peer technology is not yet mature and does not offer satisfactory and sustainable solutions to the scalability issues on all its aspects. We are concerned in this paper with the efficient retrieval of documents in a peer-to-peer network.

In general, peer-to-peer can be divided into two main classes according to its data organization, namely structured and unstructured architecture. In the structured peer-to-peer architecture such as [1–3], peers are connected to each other and resources or the records are moved to a peer according to some mapping techniques. For example, Chord employs a hashing technique to map resources to a key and peers to a key range. Peers host resources whose keys fall into their key range. In the unstructured peer-to-peer architecture [4, 5], peers are connected to each other as they join the network and resources are not being moved to other peers but hosted on site. Hence, searching for a resource requires to either broadcast the request or use routing information.

Searching strategies that have been proposed so far for most peer-to-peer architectures do not take into account the geography and the load of the physical network. Most of the current proposed peer-to-peer systems use application-level hops when measuring the performance of their searching mechanism. While the hop measurement is straightforward and easy to understand, it does not guarantee reflect actual routing time. This is due to the various factors that affect the routing time such as the physical network topology, its bandwidth, and local processing power available. For example, a hop between two peers that reside on a local area network usually take lesser time than two peers that reside on a wide area network. Furthermore, a slower machine or a popular peer might cause delay in the actual transmission time due to a longer queue. We will employ actual time as our performance measurement.

Our contribution in this paper is twofold: a general approach to indexed routing, and the presentation and comparative evaluation of three strategies to prune the information to be maintained. We first propose in this paper a solution to the creation and maintenance of indices based on the adaptive Q-routing algorithm. The algorithm maintains at each peer a table, we call the index, of the estimates of the time to reach a given resource from this peer using reinforcement learning.

Yet, given the large amount of resources to index, it is unrealistic to assume that every peer can index every resource. Therefore we propose three strategies that allow pruning of the index according to the topology of the network of peers and the popularity of the resources, respectively. We empirically analyze the respective performance of the three strategies and compare them. We show that popularity is the best criterion.

2 Related Works

In this section, we look at some of the structured (Tapestry [6], Chord [1] and CAN [2]) and unstructured (Gnutella [4] and Freenet [5]) decentralized peer-to-peer systems. Tapestry provides location-independent routing of message directly to the closest copy of an object or service using only point-to-point links and without centralized resource. Tapestry mechanisms are modeled based on the Plaxton [7] scheme; naming, structuring, core locating and routing, but they provides adaptability, fault-tolerance, and introspective optimizations. All Tapestry operations require to contact O(logN) nodes except for deleting node that requires $O(\log_2 N)$ nodes to be contacted.

Chord is a distributed lookup protocol that stores (key, value) pair for distributed data items. Each data item will be associated with a key using hash function and will be stored at the node to which key mapped. Each key k is stored on the first node whose identifier id is equal to or follows k in the identifier space. Each node maintains a routing table with information for only about O(logN) nodes. With a high probability, the number of nodes that must be contacted to resolve a query in an N-node network is O(logN).

CAN is a distributed hash-based infrastructure that provides fast lookup functionality on internet-like scales. It maintains virtual space based on "d-torus". The virtual space is partitioned into many small d-dimensional zones. Keys are mapped into zones and a zone will be split whenever a node requests to join that zone. Routing is performed by forwarding requests to the regions closest to the position of the key. In this way the expected number of hops for a search is $O(N^{1/d})$, where d is the chosen dimensionality.

Gnutella system is a decentralized file-sharing system whose participants form a virtual network communicating in peer-to-peer fashion via the Gnutella protocol which consists of 5 basic message types: ping, pong, query, queryHit and push. These messages are routed to their destination using constrained broadcast by decrementing the message's time-to-live field. A simplified HTTP GET interaction will be run to retrieve the file after requestor received location of the file from another peers.

Freenet, an anonymous peer-to-peer storage system, is decentralized, symmetric and automatically adapts when host leave and join. Freenet does not assign responsibility for documents to specific servers; instead, its lookups take the form of searches for cached copies. This allows Freenet to provide a degree of anonymity, but prevents it from guaranteeing retrieval of existing documents or from providing low bounds on retrieval costs.

In [8], the authors introduced the concept of Routing Indices (RI) in unstructured peer-to-peer network. The RI allows nodes to forward queries to the neighbors that are likely to have the needed answers. When a query arrived at a node, the node will use the RI to select a neighbor nodes and forward the query to it. The authors proposed 3 different types of RI: Compound RI, Hop-count RI and Exponential RI. Compound RI provides number of resources that can be obtained if a query were to be forwarded to a selected neighbor. While Compound RI provides only information about the quantity of resources, it provides no distance information. Hop-count RI added the distance information to the RI. Exponential RI also provides the two information but present it in a different way from Hop-count RI. The RI are constructed by propagating local information to neighboring peers at a fixed interval or when the changes in the local indices have reached a certain threshold. In the proposed scheme, resources in the network is classified into different classes and indexing is done with these classes. While this reduces the size of the index, it prevents finer granularity.

3 Background

3.1 Peer-to-peer Search and Routing Indices

Generally, searching mechanism in a peer-to-peer system can be divided into two main components: resource locating and routing. In resource locating, given a resource id, the challenge is to locate the resource in a minimal time to yield better performance and response time. In routing, the challenge is to efficiently route a resource request to the next peer in order to achieve the minimal time.

This is usually done with the help of Routing Indices. Routing indices, which is maintained at each peer, store the cost of routing requests for resources via each of its neighbors.

There might be multiple copies of similar resources scattered in the network. Successfully locating the resources within minimal time will increase the responsiveness of the system and hence making it more reliable and usable. A straightforward solution to these challenges is a centralized management model of resource sharing such as Napster [9]. However, there are several problems with using a centralized server such as the notorious single point of failure and congestion. In additional, maintaining a unified view in single node is expensive and poses scalability problems.

In this work, we focus on routing and searching strategies in totally decentralized environment. We define the routing and searching problem in peer-to-peer networks as follows (see figure 1). We consider a set of peers, $P = p_1, ..., p_n$, and a set of resources, $R = r_1, ..., r_m$, located on the peers and identified by the set of keys $K = k_1, ..., k_m$. Given a request of the form $locate(k_I)$, i.e. a request to locate resource r_i, a peer receiving this request must make a local decision as to which of its neighbors to forward the request.

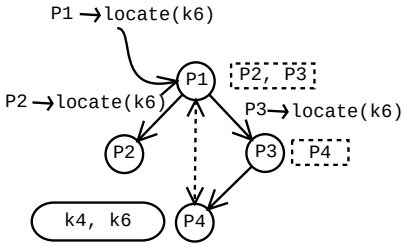

Fig. 1. An example of searching in peer-to-peer

3.2 Reinforcement Learning

In a reinforcement learning (RL) model [10], an agent learns by interacting with its environment and observing the results of these interactions. This interaction occurs as a result of the agent's capacity to perceive current environment conditions and to select the actions that should be performed in that environment. An action that is carried out changes the environment, and the agent becomes aware of these changes through a scalar reinforcement signal that is supplied by the environment. The objective of RL is to learn the mapping of states into action. The scalar reinforcement value obtained from a reinforcement function represents the system's immediate objective. The agent makes use of value function to represent the quality of the actions in the long-term. The decision to take special action when it faces a given status depends on the policy that represents the agent's

behaviour. By performing actions, and observing the resulting reward, the policy used to determine the best action for a state can be fine-tuned. Eventually, if enough states are observed an optimal decision policy will be generated and we will have an agent that performs perfectly in that particular environment. RL agent learns by receiving a reward or reinforcement from its environment, without any form of supervision other than its own decision making policy. Figure 2 depicts the general model of reinforcement learning.

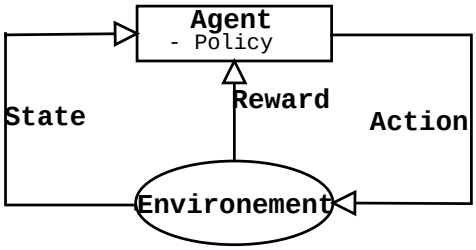

Fig. 2. Reinforcement learning model

4 Design

We propose to use an adaptive routing strategy for the routing of requests in a peer-to-peer network. This routing strategy is based on reinforcement learning: the Q-routing. At each peer, we maintain routing indices, which consist of estimates for the routing time to resources via each of the peers' immediate neighbor. We define various strategies for selecting resources to be maintained in the routing indices and analyze the startegies in the next section.

4.1 Routing Strategy

We employ Q-Routing [11] as our routing technique. Q-Routing is an adaptive routing strategy that uses Q-learning, a reinforcement learning algorithm. Q-Routing has been studied in various network routing literatures and the performance studies showed that it is able to adapt to the network traffic.

In Q-learning [12], the states and the possible actions in a given state are discrete and finite in number. It uses a set of values termed Q-values when learning. Each Q-value is denoted by *Q(s, a)* where s is a given state and a is the action taken. In general, Q-value represents the expected reinforcement yield when the action a is being taken in a state s.

Given the current state of the network, Q-Routing algorithm learns and finds an optimal routing policy from the Q-values distributed over all peers in the network. Each peer p in the network represents its own view of the state of the network through a table that stores all the Q-value, $Q_p(p', r)$, where $r \in R$, a

set of resource objects in the peer-to-peer network and $p' \in N(p)$ the set of all neighbours of peer p. $Q_p(p', r)$ is defined as the best estimated time that a query take to reach the peer that host the resource r from peer p through its neighbor p'.

The Q-value, $Q_p(p', r)$ can be represented by the following mathematical equation:

$$Q_p(p', r) = q_{p'} + \delta + Q_{p'}(p'', r) \tag{1}$$

From the equation we can derive that the minimum time needed to locate a resource r in the peer-to-peer network from the peer p through its neighbour p', is affected by 3 components:

1. The time the query spends in peer p', $q_{p'}$.
2. The transmission delay between p and p'.
3. The best estimation time for $Q_{p'}(p'', r)$.

Once we are clear what the Q-values stored at each peer stand for, it is easy to see how they can be used for making locally optimal routing decisions. When a peer p receives a query $q(s, r)$ looking for resource r, peer p will lookup its Q-value table $Q_p(*, r)$ to select the neighboring peer p' with the minimum $Q_p(p', r)$ value. With this mechanism being used in every peer in the network, the query will be answered in a minimal time.

4.2 Updates of Q-value

The performance on the routing strategy depends on all the Q-values in the path taken by the query. The closer these Q-values are to the actual time needed, the closer the routing strategy reflects the actual optimal time. Therefore, the process of updating the Q-value in a peer is very important.

Recall that when peer p receives a query $q(s, r)$, it will find a peer in the neighbour list such that Q_p fulfills the following condition:

$$Q_p(p', r) = \min_{p' \in N(p)} Q_p(p', r) \tag{2}$$

Once p' is chosen, p will then forward the query $q(s, r)$ to p'. Upon receiving the query, p' will first search if it has the requested resource r. If the requested r is not in the shared list of p', it will have to forward the query to its neighbour. For both cases, p' will have to send the Q-value back to p. In the first case, since the query can be answered by p', no further routing of the query $q(s, r)$ is needed and the Q-value returned to p is 0. While in the second case, the Q-value, $Q_{p'}(p'', r)$, returned will be according to equation 2.

The Q-value returned by p' is essentially its estimation of how long a query will be answered after p' has processed the query. The estimation returned is excluded the queuing time at p' and the transmission delay, δ, for the query to travel from p to p'. Based on the estimated value, peer p will calculate the new Q-value for $Q_p(p', r)$ with the following equation:

$$Q_p(p', r)^{new} = Q_p(p', r)^{old} + l(Q_{p'}(p'', r)^{new} + q_{p'} + \delta - Q_{p'}(p'', r)^{old}) \tag{3}$$

where l is the learning rate with $0 \leq l \leq 1$. When l is set to 1, the equation will become:

$$Q_p(p', r) = Q_{p'}(p'', r) + q_{p'} + \delta \tag{4}$$

However, since the new Q-value send back by p' is the propagation of estimated values, the Q-value is not necessary accurate. Therefore, in order to reflect the possibly error caused by estimation in the learning process, the learning rate should always be set to a value less than 1 {eg. 0.7 has been a suggested learning rate on previous network routing studies}.

4.3 Pruning of Routing Indices

Figure 3 depicts the structure of the Routing Indices (RI) that store Q-values.

Resources	Neighbour Q-value			
	p1	p2	...	pn
r1	12	6	...	8
r2	10	12	...	7
r3	4	10	...	9
rn	3	4		9

Fig. 3. Q-value table for peer p

Each resource r is a record row in the RI. For each resource, there will be corresponding Q-value for each neighbor. Since the number of resources in the network determine the number of rows, therefore, the size of the RI is proportional to the resources shared in the network and it can be potentially big. In this work, we study 3 mechanisms to maintain the size of the Q-value table:

- Stores only n entries that are closest to peer p
- Stores only n entries that are most far-off from peer p
- Stores only n that are most requested through peer p

Near-Method In near-method, we prune the routing indices according to the proximity. Initially we set the capacity of the routing indices in terms of number of entries, n, that can be stored by a peer. We prune the routing indices and store only the first n entries that have the smallest value. This method allows us to index only those resources that are closest to the peer. When the routing indices exceeded n, we remove the most distant resource in the routing indices. With this, when a peer receives a query requesting for resources that is close to the peer, the peer will be able to route the query to the neighbor that is able to answer the query within a minimal time.

Far-Method In far-method, we keep the resources in the routing indices that are furthest from the peer. Similar to near-method, we keep only maximum of n entries in the routing indices. Once routing indices overflowed, we remove the resource entry that are nearest to the peer. This allows us to index the resources that are furthest from a peer, giving more efficient routing when answering a request especially for distant resources.

Popularity-Method The third pruning method is according to popularity of resources. This method index resources that are most requested in local view. In this work, we assume the knowledge of the popular resource beforehand in order to study this pruning method. With the popularity knowledge, we apply the learning on the popular resources. Those resources that are not listed in the popularity list will be ignored.

5 Performance Analysis

We simulate the peer-to-peer network to study the performance of Q-routing and the three proposed strategies for pruning the routing indices.

The network is a randomly re-wired ring lattice. A ring lattice is a graph with n vertices, each of which is connected to its nearest k neighbors. In our experiments, we use a k value of 4. The rewiring is done by randomly choosing m vertices to connect with each other. In our experiments, we use an m value of 10% of n. This relatively small number compared to n is meant to reflect the small world effect [13].

We randomly assign the d resources to the n peers according to a uniform distribution.

The workload is determined by a Zipfian distribution corresponding to the popularity of the resources. Requests for a resource are generated according to this distribution. Requests originate from a random peer chosen uniformly.

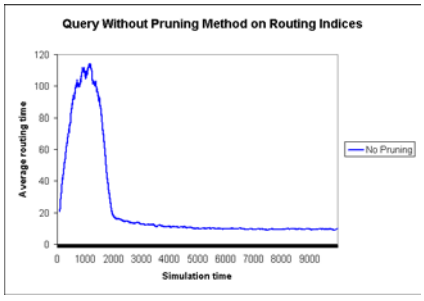

Fig. 4. Query result with Q-Routing

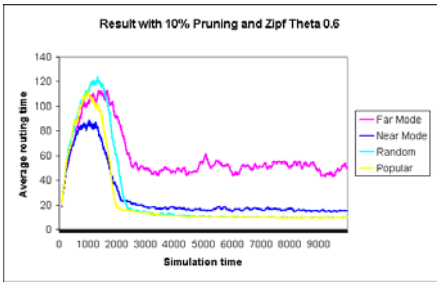

Fig. 5. 10% Pruning with Zipf θ 0.6

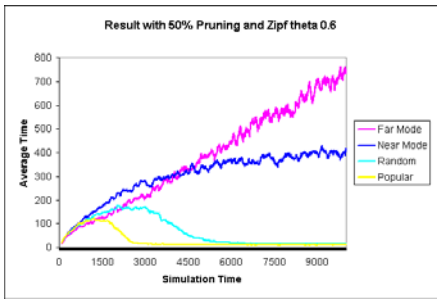

Fig. 6. 50% Pruning with Zipf θ 0.6

Fig. 7. 70% Pruning with Zipf θ 0.6

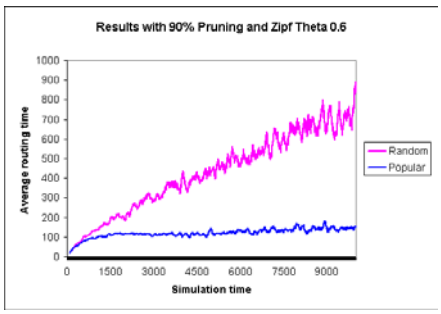

Fig. 8. 90% Pruning with Zipf θ 0.6

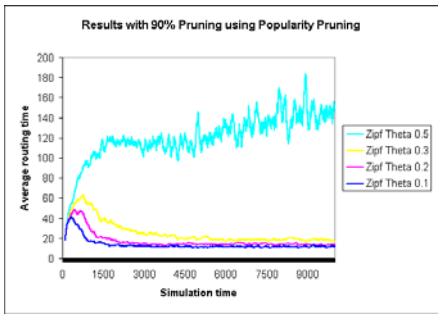

Fig. 9. 90% Popularity Pruning for various Zipf θ

Figure 4 shows the average routing time of the requests reaching destination at each unit of simulation time for a strategy using routing indices containing an entries for each resource and using Q-routing to estimate the cost of reaching the resource. The result of this experiment confirms the adaptive property of Q-routing. Indeed, after an initial unstable learning phase, the average routing time continuously converges toward a minimum.

The result of this experiment confirms the adaptive property of Q-routing. Indeed, after an initial unstable learning phase, the average routing time continuously converges toward a minimum.

Figures 5, 6, 7, 8, and 9 show the average routing time of the requests reaching destination at each unit of simulation time for several strategies using pruned routing indices.

In Figures 5, 6, 7, and 8, we first compare the performance of the near-method, the far-method, the popularity-method as well as, for reference, the performance of a method randomly pre-selecting the resources to be kept in every routing index. For these three figures, the popularity of the resources is determined by a Zipfian with a θ coefficient of value 0.6. We fix the number of entries pruned from the routing indices to 10%, 50%, 70%, and 90%, for each figure respectively.

We only present the performance of the near- and far- methods on the first two figures since it immediately appears that these methods do not scale up as they cannot route efficiently with 50% and less of the documents in the routing indices. This phenomenon is not only due to the possible irrelevance of the entries in the routing indices but also to their high instability and therefore to the inaccuracy of the content of the routing indices under these two approaches. Indeed the two approaches are outperformed by the random-method. The popularity-method, on the contrary, continues to outperform the random-method and to stabilize to a reasonable routing time up to 90% pruning.

Figure 9 shows the average routing time of the requests reaching destination at each unit of simulation time for the popularity-method with a varying skewedness of the popularity distribution. We control this aspect by choosing different values of the θ parameter. The smaller θ, the skewer the distribution. We present the results for four values of θ: 0.5, 0.3, 0.2, and 0.1. Skewer popularity improves the convergence of the popularity-method. Indeed as there are fewer more popular documents, more requests find appropriate and more accurate entries in the routing indices.

6 Conclusion and Future Works

We have presented a routing strategy for requests in peer-to-peer networks based on adaptive routing in routing indices Q-routing. The strategy uses reinforcement learning to estimate the cost of routing a request. Yet such a strategy is scalable only if the routing indices are of reasonable size. It is not realistic to consider maintaining at each peer a complete routing index of all the resources in the network. We proposed and comparatively evaluated three methods for the pruning for the pruning of the routing indices.

Our experiments confirm the validity of the adaptive routing and the scalability of a pruning approach based on popularity of the resources.

Our analysis, however, relies on a predetermined popularity. We are currently extending this work and devising a strategy in which peers adaptively learn the resource popularity based on the requests they route. In such a strategy, reinforcement learning is used twice: for estimating the cost of routing, and for estimating the popularity, forward, and backward, respectively. Such a method would constitute a complete, effective, and efficient routing solution for unstructured peer-to-peer networks

References

1. D. Karger F. Kaashoek I. Stoica, R. Morris and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.
2. M. Handley R. Karp S. Ratnasamy, P. Francis and S. Shenker. A scalable content addressable network. In *Proceedings of the 2001 ACM SIGCOMM Conference*, 2001.

3. P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, 2001.
4. GNUTELLA. http://gnutella.wego.com.
5. B. Wiley I. Clarke, O. Sandberg and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Lecture Notes in Computer Science*, 2001.
6. J. D. Kubiatowicz B. Y. Zhao and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.
7. C. Plaxton R. Rajaraman and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 311–320, 1997.
8. J. Kubiatowicz et al. Routing indices for peer-to-peer systems. In *Proceedings International Conference on Distributed Computing Systems*, July 2002.
9. NAPSTER. http://www.napster.com.
10. Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. In *Journal of Artificial Intelligence Research*, volume 4, pages 237–285, 1996.
11. Michael Littman and Justin Boyan. A distributed reinforcement learning scheme for network routing. In *Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications*, 1993.
12. Sutton R. S. and Barto A. G. *Reinforcement learning: An introduction*. MIT Press, 1998.
13. A. Oram, editor. *Peer-to-Peer : Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, 2001.